



# Desarrollo Seguro en Python

## Lección 4: Auditoría y Securización de Código

# ÍNDICE

<b>Auditoría y Securización de Código .....</b>	<b>2</b>
<b>1. Introducción .....</b>	<b>3</b>
<b>2. Seguridad en Proyectos python.....</b>	<b>3</b>
<b>3. Bandit .....</b>	<b>5</b>
<b>4. Pyflakes .....</b>	<b>6</b>
<b>5. PyLint.....</b>	<b>7</b>
<b>6. SonarQube.....</b>	<b>9</b>
6.1 Funcionamiento .....	9
6.2 Instalación.....	10
6.3 Analizando un proyecto.....	12
6.4 Ejemplo práctico .....	13
<b>7. Puntos clave .....</b>	<b>20</b>

# Auditoría y Securización de Código



## *Objetivos*

- Conocer las vulnerabilidades conocidas del lenguaje con el que estamos trabajando.
- Aplicar herramientas para auditar y securizar código.
- Saber cómo proceder para securizar nuestro código.

## 1. INTRODUCCIÓN

Llegados a este punto ya conocemos las principales listas de vulnerabilidades que puede tener el software y los principales controles proactivos que podemos aplicar para desarrollar un software lo más seguro posible. El siguiente paso será conocer herramientas diseñadas para analizar y auditar código en busca de posibles errores de seguridad y conocer las vulnerabilidades del lenguaje o framework con el que estamos trabajando.

## 2. SEGURIDAD EN PROYECTOS PYTHON

Un punto muy importante a tener en cuenta cuando vamos a trabajar con un lenguaje específico es analizar y estudiar sus componentes inseguros o vulnerabilidades conocidas. En este punto vamos a hacer un inciso sobre las funciones de Python y sus frameworks.

En la siguiente figura se presenta una lista de los componentes de Python inseguros más conocidos, donde podemos destacar funciones como eval, pickle y los módulos os, subprocess o yaml.

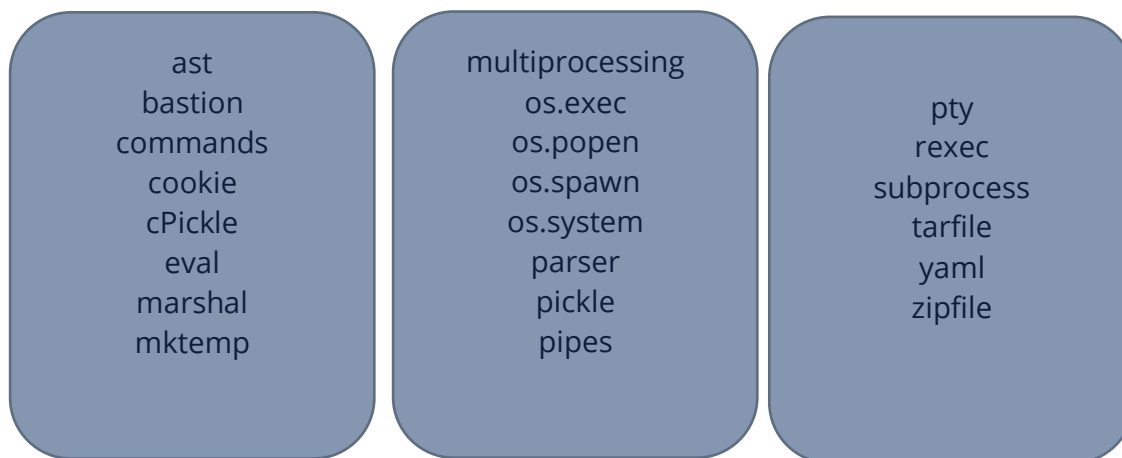


Figura 2.1. Componentes inseguros Python.

Algunos módulos de Python, como el de subprocess, si se usan de forma insegura, pueden suponer un riesgo para la aplicación. La documentación de Python normalmente incluye una nota sobre los principales riesgos si, por ejemplo, usamos el parámetro `shell=True` al intentar ejecutar un comando con la función `subprocess.call()`:

### Security Considerations

Unlike some other popen functions, this implementation will never implicitly call a system shell. This means that all characters, including shell metacharacters, can safely be passed to child processes. If the shell is invoked explicitly, via `shell=True`, it is the application's responsibility to ensure that all whitespace and metacharacters are quoted appropriately to avoid [shell injection vulnerabilities](#).

When using `shell=True`, the `shlex.quote()` function can be used to properly escape whitespace and shell metacharacters in strings that are going to be used to construct shell commands.

Figura 2.2 Consideraciones de seguridad Python 3.9.

<https://docs.python.org/3.9/library/subprocess.html#security-considerations>

Entre los principales problemas relacionados con la seguridad, podemos destacar:

- Uso de funciones Python peligrosas, como `eval()`, sin la correcta validación de las cadenas de entrada.
- Acceso al sistema de archivos.
- Los fragmentos de código SQL y JavaScript integrados en el código fuente o las plantillas, especialmente si estamos utilizando Django o Flask. En este punto, se recomienda seguir las mejores prácticas usando mecanismos de persistencia como Django ORM: <https://docs.djangoproject.com/en/3.1/topics/db/> para persistir en las plantillas de bases de datos y evitar la inyección de SQL y XSS.
- Claves API hardcodeadas en el código fuente.
- Llamadas HTTP a servicios web internos o externos.
- Flask-Security <https://pythonhosted.org/Flask-Security/>

### 3. BANDIT



Bandit es una herramienta diseñada para encontrar problemas de seguridad comunes en el código Python. Para hacer esto, Bandit procesa cada archivo, crea un AST (Árbol de Sintaxis Abstracta) a partir de él y ejecuta los complementos apropiados contra los nodos del árbol. Una vez que Bandit ha terminado de escanear todos los archivos, genera un informe.

Bandit se desarrolló originalmente dentro del OpenStack Security Project y luego se reubicó en PyCQA. <https://pypi.org/project/bandit/>

Mejores formas para instalar Bandit

Creando un entorno virtual

```
virtualenv bandit-env
# Or if you're working with a Python 3 project
python3 -m venv bandit-env
# And activate it:
source bandit-env/bin/activate
```

Figura 3.1 Entorno virtual para Bandit en Python.

Instalar Bandit:

```
pip install bandit
# Or if you're working with a Python 3 project
pip3 install bandit
```

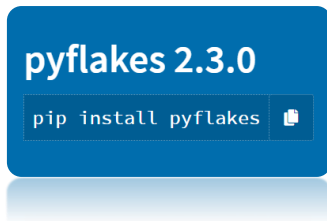
Figura 3.2 Instalación de Bandit.

Ejecutar Bandit:

```
bandit -r path/to/your/code
```

Figura 3.3 Ejecución de Bandit.

## 4. PYFLAKES

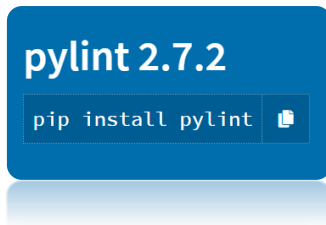


Permite analizar el código de Python en busca de errores. Si lo comparamos con otras herramientas de estilo, como PyLint o Pychecker, Pyflakes analiza de forma más rápida, ya que solo examina el árbol de sintaxis de cada archivo de forma individual.

<https://pypi.org/project/pyflakes/>

El código fuente se encuentra disponible en el repositorio GitHub <https://github.com/PyCQA/pyflakes> y se puede instalar desde el repositorio oficial de python mediante el comando pip install y resulta compatible con todas las versiones de Python.

## 5. PYLINT



PyLint <https://pylint.org> tiene como objetivo analizar código en Python en busca de errores o síntomas de mala calidad en el código fuente. Cabe destacar que, por defecto, la guía de estilo que trata de seguir es la descrita en PEP-8 <https://www.python.org/dev/peps/pep-0008/>

Entre las revisiones básicas que realiza, podemos destacar:

- Presencia de cadenas de documentación (docstring)
- Nombres de módulos, clases, funciones, métodos, argumentos y variables.
- Número de argumentos, variables locales, retornos y sentencias en funciones y métodos
- Atributos requeridos para módulos
- Valores por omisión no recomendados como argumentos
- Redefinición de funciones, métodos y clases
- Uso de declaraciones globales

En cuanto a la revisión de variables, es capaz de detectar los siguientes casos:

- Determinación de si una variable o import no está siendo usado
- Detección de variables indefinidas
- Redefinición de variables provenientes de módulos builtins o de ámbito externo
- Uso de una variable antes de asignación de valor

Entre las revisiones de clases, podemos destacar:

- Métodos sin self como primer argumento
- Acceso único a miembros existentes vía self
- Atributos no definidos en el método `__init__`
- Código inalcanzable



Entre las revisiones de diseño, podemos destacar:

- Número de métodos, atributos y variables locales
- Tamaño, complejidad de funciones y métodos

En las revisiones de importaciones, podemos destacar:

- Dependencias externas
- Imports relativos o imports de todos los métodos y variables vía `*(wildcards)`.
- Uso de imports cíclicos
- Uso de módulos obsoletos.

Otras revisiones:

- Código fuente con caracteres ASCII sin tener una declaración de `encoding`.
- Búsqueda por similitudes o duplicación en el código fuente.
- Revisión de excepciones.
- Revisiones de tipo haciendo uso de inferencia de tipos.

Con posterioridad a los mensajes de análisis mostrados, PyLint genera una serie de reportes, cada uno de ellos enfocándose en un aspecto particular del proyecto, como número de mensajes por categorías y las dependencias de módulos.

## 6. SONARQUBE



SonarQube es una plataforma para evaluar código fuente. Es software libre y usa diversas herramientas de análisis estático de código fuente como Checkstyle, PMD o FindBugs para obtener métricas que pueden ayudar a mejorar la calidad del código de un programa.

### 6.1 Funcionamiento

El servidor SonarQube ejecuta los siguientes procesos:

- un servidor web que sirve a la interfaz de usuario de SonarQube.
- un servidor de búsqueda basado en Elasticsearch.
- el motor de cálculo encargado de procesar los informes de análisis de código y guardarlos en la base de datos de SonarQube.

La base de datos para almacenar lo siguiente:

- Métricas y problemas de calidad y seguridad del código generados durante los escaneos de código.
- La configuración de la instancia de SonarQube.
- Uno o más escáneres que se ejecutan en sus servidores de integración continua o de compilación para analizar proyectos.

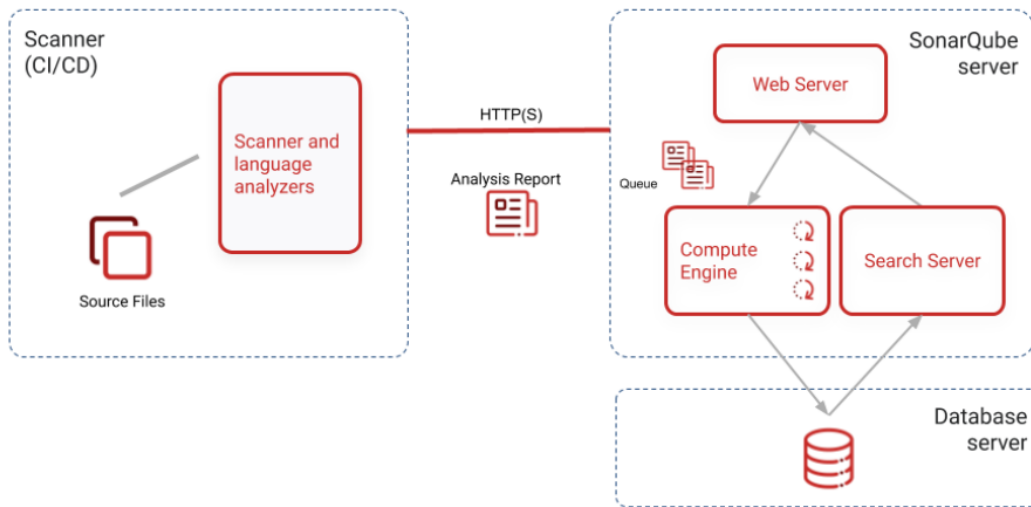


Figura 6.1 Esquema de componentes de SonarQube.

## 6.2 Instalación

Vamos a realizar una instalación de una instancia local. Esto podemos hacerlo descargando el fichero zip con SonarQube o desde Docker.

[Descarga SonarQube](#)

### ▼ From the zip file

1. [Download](#) the SonarQube Community Edition zip file.
2. As a **non-root user**, unzip it, let's say in `C:\sonarqube` or `/opt/sonarqube`.
3. As a **non-root user**, start the SonarQube Server:

```
# On Windows, execute:
C:\sonarqube\bin\windows-x86-64\StartSonar.bat

# On other operating systems, as a non-root user execute:
/opt/sonarqube/bin/[OS]/sonar.sh console
```

**i** If your instance fails to start, check your [logs](#) to find the cause.

Figura 6.2 Instalación SonarQube desde zip

Para poder ejecutar esta versión necesitamos java 11 en nuestra máquina.

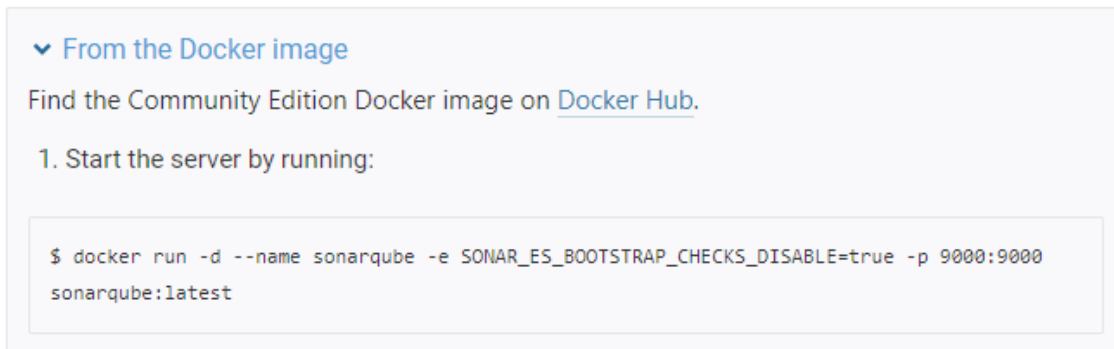


Figura 6.3 Instalación desde Docker

Una vez que la instancia esté en funcionamiento, iniciamos sesión en <http://localhost:9000> con las credenciales de administrador del sistema:

- inicio de sesión: admin
- contraseña: admin

SonarScanner

[Descarga SonarScanner](#)

SonarScanner es el escáner que se utiliza cuando no hay un escáner específico para su sistema de compilación.

## 6.3 Analizando un proyecto

Una vez que nos hemos logueado dentro del sistema vamos a analizar un proyecto.

1. Hacer clic en el botón Crear nuevo proyecto.
2. Asigne a su proyecto una clave de proyecto y un nombre para mostrar y haga clic en el botón Configurar.
3. En Proporcionar un token, seleccione Generar un token. Dale un nombre a tu token, haz clic en el botón Generar y haz clic en Continuar.
4. Seleccione el idioma principal de su proyecto en Ejecutar análisis en su proyecto y siga las instrucciones para analizar su proyecto. Aquí descargará y ejecutará un escáner en su código (si está utilizando Maven o Gradle, el escáner se descarga automáticamente).

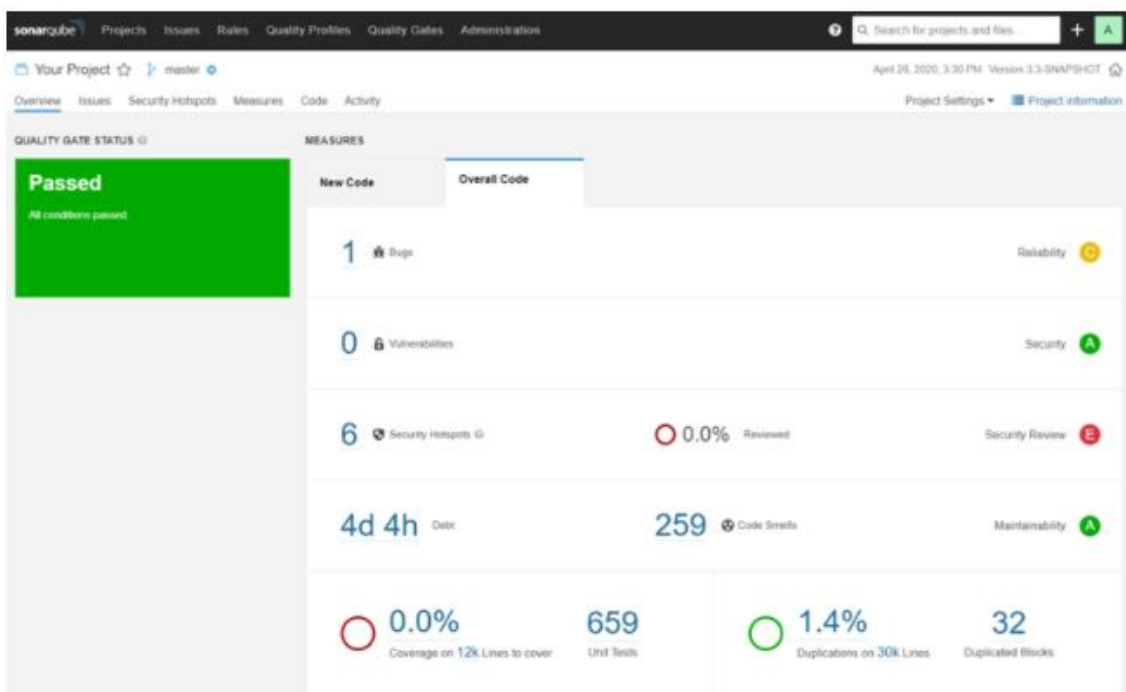


Figura 6.4 Captura de informe SonarQube.

## 6.4 Ejemplo práctico

Vamos a auditar el proyecto Veotek. Proyecto de código abierto diseñado para registrar las actividades realizadas dentro de las instalaciones de la empresa. Obtener reportes por personal, fecha y mes. Además, se obtiene el registro de las asistencias a la empresa de todos los empleados.

En primer lugar, descargamos el proyecto y posteriormente arrancamos SonarQube.

```
c:\ciberseguridad\sonarqube\bin\windows-x86-64  
λ StartSonar.bat |
```

Figura 6.5 Arrancamos instancia local de SonarQube

Accedemos a la instancia local de SonarQube con nuestras credenciales.

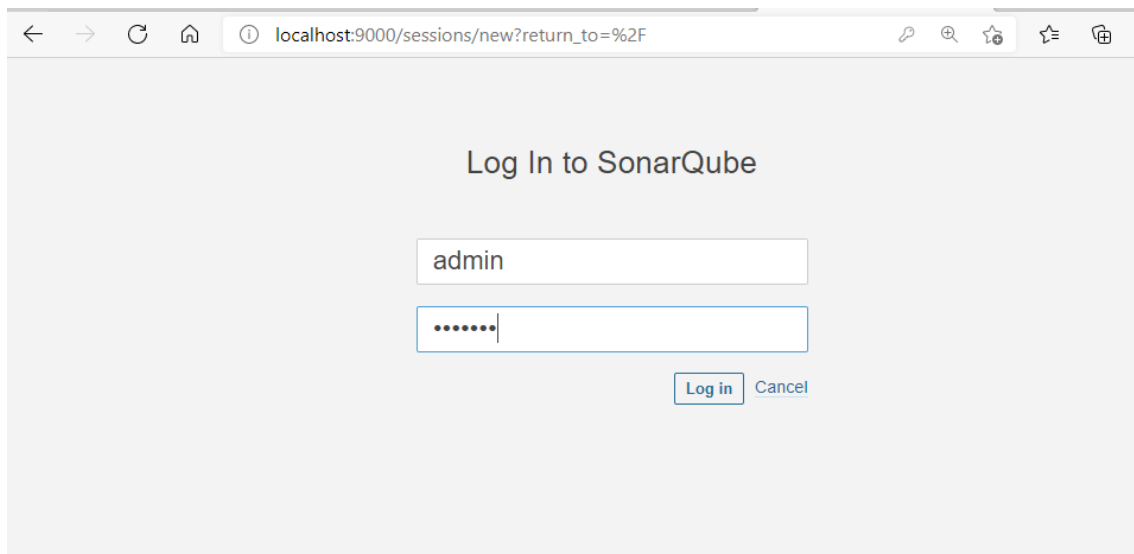


Figura 6.6 Accedemos al servidor local.

Creamos un proyecto nuevo y asignamos la key del proyecto.

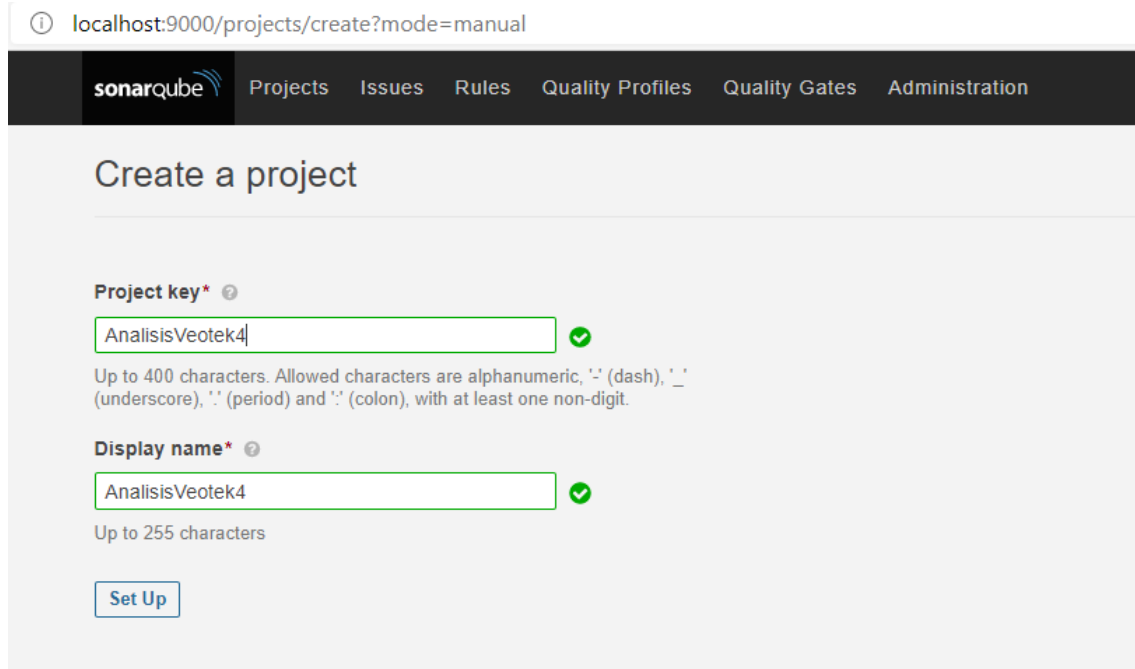


Figura 6.7 Creamos nuevo proyecto.

Generación de un token para nuestro proyecto.

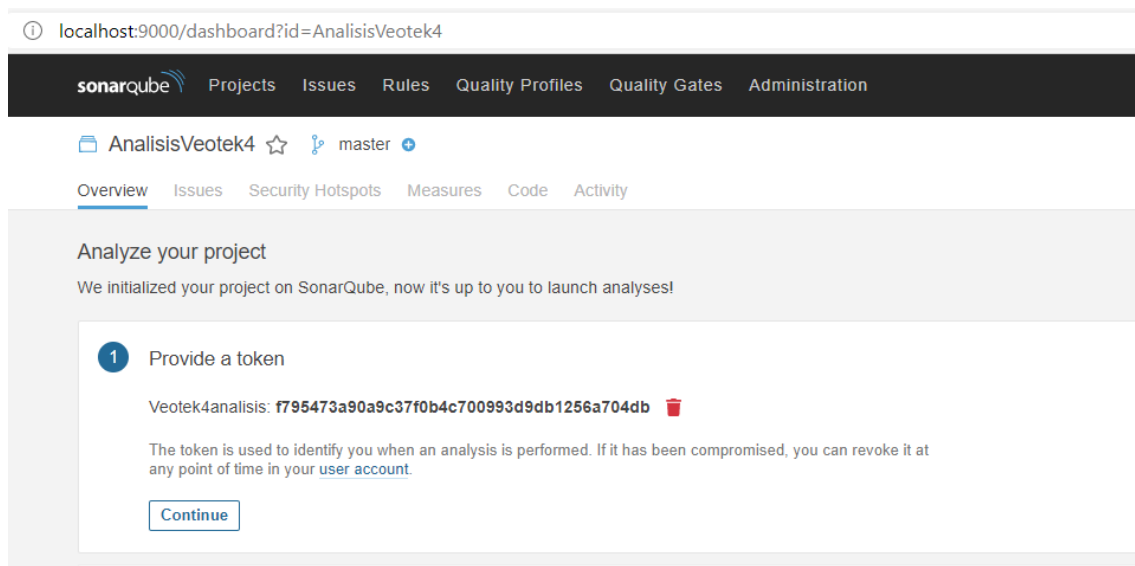


Figura 6.8 Creamos un token para nuestro proyecto.

Elegimos la tecnología y sistema operativo de nuestro proyecto. En la parte inferior nos genera la línea que debemos ejecutar para realizar el análisis de nuestro proyecto. La copiamos y lanzamos desde consola de comandos.

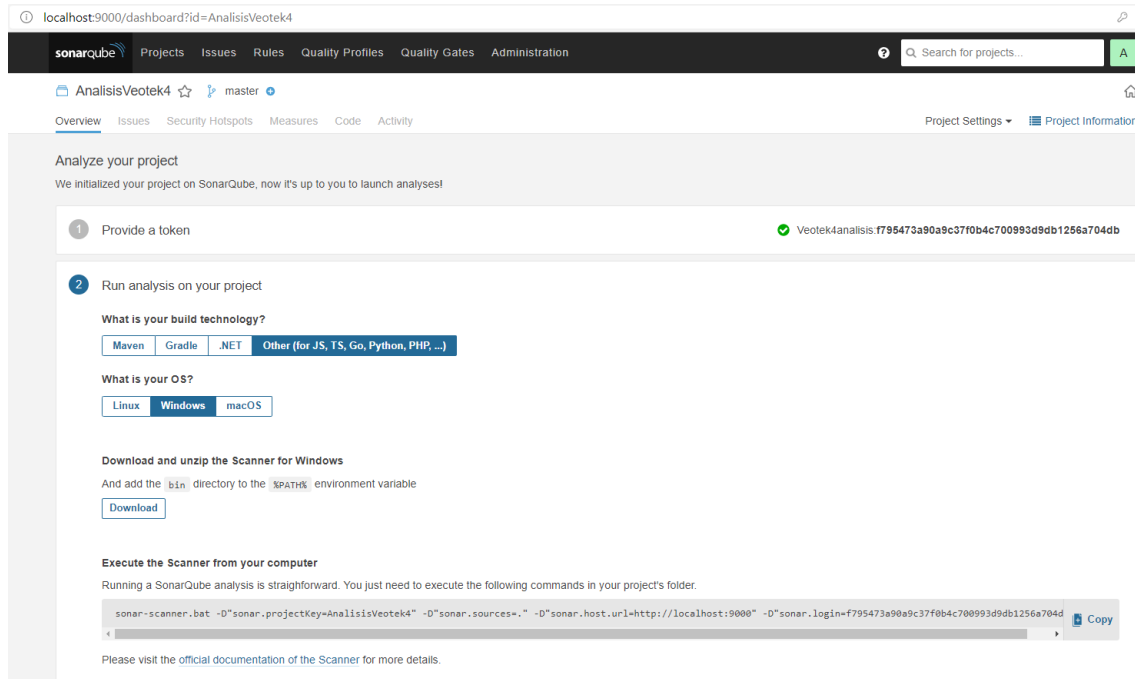


Figura 6.9 Sonar Scanner

Procedemos a lanzar el análisis dentro de la carpeta del proyecto.

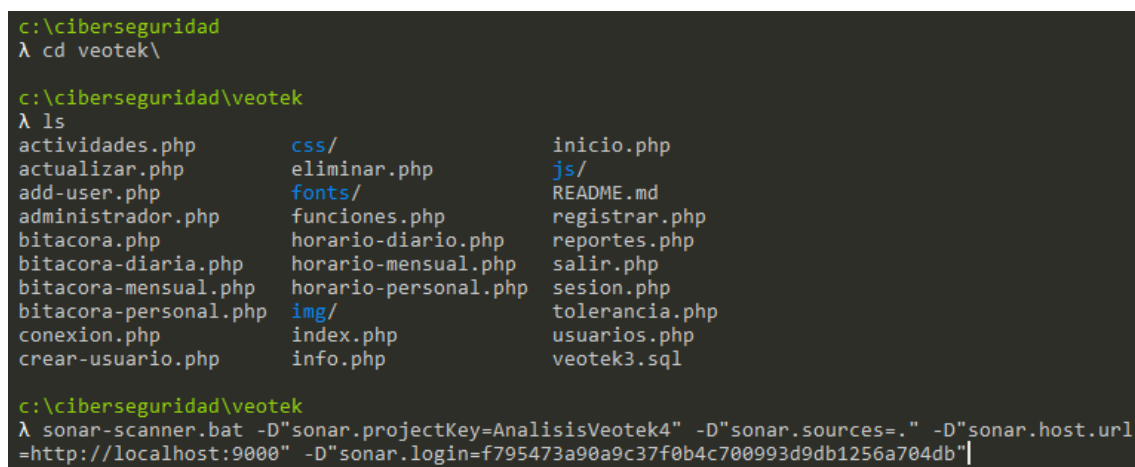


Figura 6.10 Ejecución de Sonar Scanner



Una vez que finaliza el análisis nos genera el siguiente reporte automáticamente en la instancia local de SonarQube.

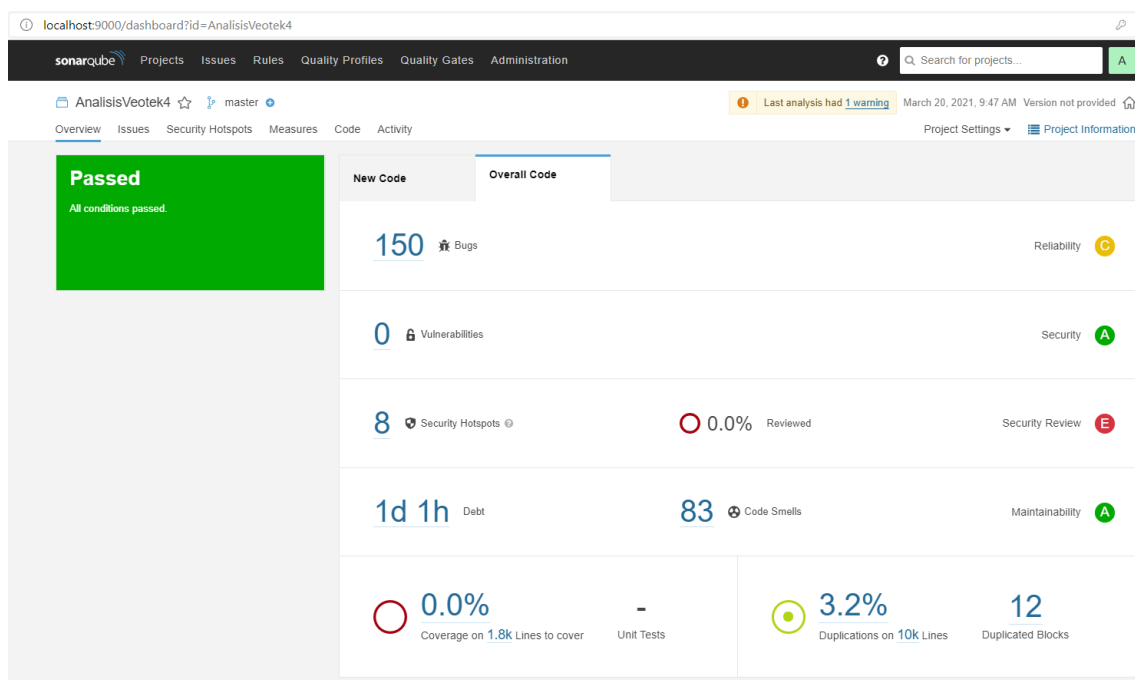


Figura 6.11 Reporte de Sonar Scanner

En los reportes hay que tener en cuenta:

- Bugs: errores de programación
- Vulnerabilidades: son errores que afectan a la seguridad.
- Hotspots: fragmentos de código relacionados con la seguridad, que deben ser manualmente revisados. Permiten solucionar problemas de seguridad y nos ayudan a aprender sobre seguridad
- Code Smells: malas prácticas de programación que dificultan que el código pueda mantenerse

En relación a calidad del código también es importante tener en cuenta el porcentaje de código duplicado.

Vamos a analizar los Hotspots:

The screenshot shows the SonarQube web interface. The top navigation bar includes links for Projects, Issues, Rules, Quality Profiles, Quality Gates, and Administration. The main content area displays a security hotspot for the project 'AnalysisVeotek4'. The hotspot is titled 'Review this hardcoded credential.' and is categorized under 'Authentication'. The review priority is 'HIGH'. The status is 'To review'. The code snippet shows a PHP connection string: `$conexion = mysql_connect("localhost", "root", "veotek");`. The risk analysis section explains that hard-coded credentials are a security risk and provides recommendations for storing credentials in configuration files or using a management service for secrets.

Figura 6.12 Análisis de HotSpots

En la parte de la izquierda nos los clasifica por prioridades, en la descripción del error nos muestra la categoría, líneas y fichero afectado. En la parte inferior nos da consejos e información para poder subsanarlo. ¿Cuál es el riesgo? ¿Estoy en un riesgo? ¿Cómo puedo solucionarlo?

Si analizamos la información que nos da primero de los riesgos vemos que lo primero que nos dice son las CVE (Common Vulnerabilities and Exposures) con las que se relaciona.

What's the risk?	Are you at risk?	How can you fix it?
<p>Because it is easy to extract strings from an application source code or binary, credentials should not be hard-coded. This is particularly true for applications that are distributed or that are open-source.</p> <p>In the past, it has led to the following vulnerabilities:</p> <ul style="list-style-type: none"> <li>• <a href="#">CVE-2019-13466</a></li> <li>• <a href="#">CVE-2018-15389</a></li> </ul> <p>Credentials should be stored outside of the code in a configuration file, a database, or a management service for secrets.</p> <p>This rule flags instances of hard-coded credentials used in database and LDAP connections. It looks for hard-coded credentials in connection strings, and for variable names that match any of the patterns from the provided list.</p> <p>It's recommended to customize the configuration of this rule with additional credential words such as "oAuthToken", "secret", ...</p>		

Figura 6.13 Relación con CVE

En la siguiente pestaña ¿Estás en riesgo? Nos lanza una serie de cuestiones para que analicemos nuestro código y nos da ejemplos de código.

What's the risk?	Are you at risk?	How can you fix it?
<p>Ask Yourself Whether</p> <ul style="list-style-type: none"> <li>• Credentials allows access to a sensitive component like a database, a file storage, an API or a service.</li> <li>• Credentials are used in production environments.</li> <li>• Application re-distribution is required before updating the credentials.</li> </ul> <p>There is a risk if you answered yes to any of those questions.</p> <p>Sensitive Code Example</p> <pre>\$password = "65DBGgwe4uazdwQA"; // Sensitive  \$httpUrl = "https://example.domain?user=user&amp;password=65DBGgwe4uazdwQA" // Sensitive \$sshUrl = "ssh://user:65DBGgwe4uazdwQA@example.domain" // Sensitive</pre>		

Figura 6.14 ¿Estás en riesgo?

En la pestaña de ¿Cómo puedo solucionarlo? Nos da recomendaciones para prácticas seguras de código.


```

1 <?php
2 $conexion = mysql_connect("localhost", "root", "veotek");
3 mysql_select_db("veotek3",$conexion);
4 ?>

```

What's the risk?	Are you at risk?	How can you fix it?
<p><b>Recommended Secure Coding Practices</b></p> <ul style="list-style-type: none"> <li>• Store the credentials in a configuration file that is not pushed to the code repository.</li> <li>• Store the credentials in a database.</li> <li>• Use your cloud provider's service for managing secrets.</li> <li>• If the a password has been disclosed through the source code: change it.</li> </ul> <p><b>Compliant Solution</b></p> <pre> \$user = getUser(); \$password = getPassword(); // Compliant  \$httpUrl = "https://example.domain?user=\$user&amp;password=\$password" // Compliant \$sshUrl = "ssh://\$user:\$password@example.domain" // Compliant </pre> <p><b>See</b></p> <ul style="list-style-type: none"> <li>• <a href="#">OWASP Top 10 2017 Category A2</a> - Broken Authentication</li> <li>• <a href="#">MITRE, CWE-798</a> - Use of Hard-coded Credentials</li> <li>• <a href="#">MITRE, CWE-259</a> - Use of Hard-coded Password</li> <li>• <a href="#">CERT, MSC03-J</a> - Never hard code sensitive information</li> <li>• <a href="#">SANS Top 25</a> - Porous Defenses</li> <li>• Derived from FindSecBugs rule <a href="#">Hard Coded Password</a></li> </ul>		

Figura 6.15 ¿Estás en riesgo?



**Importante**

En la parte inferior nos hace mención a:

- OWASP
- CWE
- CERT
- SANS
- FIND SECURITY BUS

Ahora con toda esta información si realmente los hemos identificado como un riesgo procederemos a corregirlo en función de las recomendaciones que nos da la herramienta.

## 7. PUNTOS CLAVE

- | Conocer herramientas que nos permitan auditar nuestro código buscando bugs y problemas de seguridad es un paso más en el camino de la Securización de nuestras aplicaciones. Hemos de utilizar siempre herramientas de análisis lo más actualizadas posibles y con una gran comunidad detrás.
- | Estas herramientas de auditoria junto con buenas prácticas de desarrollo seguro aplicando controles proactivos y una cultura de seguridad en la empresa y en el ciclo de desarrollo de nuestras aplicaciones permitirá dar un mayor nivel de seguridad y calidad a nuestros desarrollos.

